**Template**

**1并查集**

int find(int x) { return fa[x] == x ? x : fa[x] = find(fa[x]); }

**若启发式合并：**

x, y : root

if(rank[x] < rank[y]) fa[x] = y;

else { fa[y] = x; if(rank[x] == rank[y]) rank[x]++; }

**2树状数组**

int n, c[N];  
  
int lowbit(int x) { return x & (-x); }  
  
int sum(int x) {  
 int result = 0;  
 while(x > 0) {  
 result += c[x];  
 x -= lowbit(x);  
 }  
 return result;  
}  
  
void add(int x, int d) {  
 while(x <= n) {  
 c[x] += d;  
 x += lowbit(x);  
 }  
}

**3 RMQ**

int n, a[N], d[N][15];  
  
void RMQ\_init(int n) {  
 for(int i = 0; i < n; i++) d[i][0] = a[i];  
 for(int j = 1; (1 << j) <= n; j++)  
 for(int i = 0; i + (1 << j) <= n; i++)  
 d[i][j] = min(d[i][j - 1], d[i + (1 << (j - 1))][j - 1]);  
}  
  
int RMQ(int l, int r) {  
 int k = 0;  
 while((1 << (k + 1)) <= r - l + 1) k++;  
 return min(d[l][k], d[r - (1 << k) + 1][k]);  
}

**4 Check质数，生成质数表**

// Generate prime numbers  
void sieve(int n) {  
 int m = (int)sqrt(n + 0.5);  
 memset(vis, 0, sizeof(vis));  
 for(int i = 2; i <= m; i++) if(!vis[i])  
 for(int j = i \* i; j <= n; j += i) vis[j] = true;  
}  
  
// Return the number of prime numbers  
int gen\_prime(int n) {  
 vector<int> prime;  
 sieve(n);  
 for(int i = 2; i <= n; i++)  
 if(!vis[i]) { prime.push\_back(i); cout << i << endl; }  
 return (int)prime.size();  
}

**5欧几里得&扩展欧几里得**

LL gcd(LL a, LL b) { return !b ? a : gcd(b, a % b); }  
  
// abs(x) + abs(y) is smallest  
void gcd(LL a, LL b, LL& d, LL& x, LL& y) {  
 if(!b) { d = a; x = 1; y = 0; }  
 else { gcd(b, a % b, d, y, x); y -= x \* (a / b); }  
}

**6 a^p mod n**

// a^p mod n  
LL pow\_mod(LL a, LL p, LL n) {  
 if(!p) return 1;  
 LL ans = pow\_mod(a, p / 2, n);  
 ans = ans \* ans % n;  
 if(p % 2) ans = ans \* a % n;  
 return ans;  
}

**7欧拉phi函数&phi表**

// Euler Phi function  
  
int euler\_phi(int n) {  
 int m = (int)sqrt(n + 0.5);  
 int ans = n;  
 for(int i = 2; i <= m; i++) if(n % i == 0) {  
 ans = ans / i \* (i - 1);  
 while(n % i == 0) n /= i;  
 }  
 if(n > 1) ans = ans / n \* (n - 1);  
 return ans;  
}  
  
int phi[N];  
void phi\_table(int n) {  
 memset(phi, 0, sizeof(phi));  
 phi[1] = 1;  
 for(int i = 2; i <= n; i++) if(!phi[i])  
 for(int j = i; j <= n; j += i) {  
 if(!phi[j]) phi[j] = j;  
 phi[j] = phi[j] / i \* (i - 1);  
 }  
}

**8 a^(-1) mod n (扩展欧几里得与费马小定理)**

int phi[N];  
void phi\_table(int n) {  
 memset(phi, 0, sizeof(phi));  
 phi[1] = 1;  
 for(int i = 2; i <= n; i++) if(!phi[i])  
 for(int j = i; j <= n; j += i) {  
 if(!phi[j]) phi[j] = j;  
 phi[j] = phi[j] / i \* (i - 1);  
 }  
}  
  
// a^(-1) mod n by Extended Euclid Algorithm  
// Since by extended Euclid, abs(x) + abs(y) is smallest  
LL inv(LL a, LL n) {  
 LL d, x, y;  
 gcd(a, n, d, x, y);  
 return d == 1 ? (x + n) % n : -1;  
}  
  
// a^(-1) mod n by Fermat Small Theorem  
LL inv\_Fermat(LL a, LL n) {  
 return gcd(a, n) == 1 ? pow\_mod(a, euler\_phi(n) - 1, n) : -1;  
}

**9线段树点修改**

int minv[2 \* N + 10], a[N], n;  
  
// minv starts with 1, same as BIT.  
// Are are closed intervals.  
  
int query(int o, int l, int r, int ql, int qr) { // query [ql, qr]. The current is [l, r].  
 int mid = l + (r - l) / 2, ans = INF;  
 if(ql <= l && r <= qr) return minv[o];  
 if(ql <= mid) ans = min(ans, query(2 \* o, l, mid, ql, qr));  
 if(qr > mid) ans = min(ans, query(2 \* o + 1, mid + 1, r, ql, qr));  
 return ans;  
}  
  
void update(int o, int l, int r, int p, int v) { // A[p] = v  
 if(l == r) minv[o] = v;  
 else {  
 int mid = l + (r - l) / 2;  
 if(p <= mid) update(2 \* o, l, mid, p, v);  
 else update(2 \* o + 1, mid + 1, r, p, v);  
 minv[o] = min(minv[2 \* o], minv[2 \* o + 1]);  
 }  
}  
  
int build(int o, int l, int r) { // Build the segment tree in O(n)  
 if(l == r) return minv[o] = a[l];  
 else {  
 int mid = l + (r - l) / 2;  
 return minv[o] = min(build(o \* 2, l, mid), build(o \* 2 + 1, mid + 1, r));  
 }  
}

**10线段树区间修改**

int minv[N \* 2], maxv[N \* 2], sumv[N \* 2], addv[N \* 2], setv[N \* 2], minm, maxm, summ;  
int a[N];  
// set的值 >= 0  
// 若addv和setv同时存在则说明是先被set后被add  
  
void push\_down(int o) {  
 if(setv[o] >= 0) {  
 setv[o \* 2] = setv[o \* 2 + 1] = setv[o];  
 addv[o \* 2] = addv[o \* 2 + 1] = 0;  
 setv[o] = -1;  
 }  
 if(addv[o] > 0) {  
 addv[o \* 2] += addv[o];  
 addv[o \* 2 + 1] += addv[o];  
 addv[o] = 0;  
 }  
}  
  
void maintain(int o, int l, int r) {  
 sumv[o] = minv[o] = maxv[o] = 0;  
 if(setv[o] >= 0) {  
 sumv[o] = setv[o] \* (r - l + 1);  
 maxv[o] = minv[o] = setv[o];  
 } else if(r > l) {  
 sumv[o] += sumv[o \* 2] + sumv[o \* 2 + 1];  
 maxv[o] = max(maxv[o \* 2], maxv[o \* 2 + 1]);  
 minv[o] = min(minv[o \* 2], minv[o \* 2 + 1]);  
 }  
 sumv[o] += addv[o] \* (r - l + 1);  
 minv[o] += addv[o];  
 maxv[o] += addv[o];  
}  
  
void add(int o, int l, int r, int ql, int qr, int v) {  
 if(ql <= l && r <= qr) {  
 addv[o] += v;  
 } else {  
 push\_down(o);  
 int mid = l + (r - l) / 2;  
  
 // Need to maintain!!!!  
 if(ql <= mid) add(o \* 2, l, mid, ql, qr, v); else maintain(o \* 2, l, mid);  
 if(qr > mid) add(o \* 2 + 1, mid + 1, r, ql, qr, v); else maintain(o \* 2 + 1, mid + 1, r);  
 }  
 maintain(o, l, r);  
}  
  
  
void set(int o, int l, int r, int ql, int qr, int v) {  
 if(ql <= l && r <= qr) {  
 setv[o] = v;  
 addv[o] = 0; // Important  
 } else {  
 push\_down(o);  
 int mid = l + (r - l) / 2;  
 if(ql <= mid) set(o \* 2, l, mid, ql, qr, v); else maintain(o \* 2, l, mid);  
 if(qr > mid) set(o \* 2 + 1, mid + 1, r, ql, qr, v); else maintain(o \* 2 + 1, mid + 1, r);  
 }  
 maintain(o, l, r);  
}  
  
void query(int o, int l, int r, int ql, int qr, int ad) {  
 if(setv[o] >= 0) { // Important to add addv  
 summ += (setv[o] + ad + addv[o]) \* (min(r, qr) - max(l, ql) + 1);  
 minm = min(minm, setv[o] + ad + addv[o]);  
 maxm = max(maxm, setv[o] + ad + addv[o]);  
 } else if(ql <= l && r <= qr) {  
 summ += sumv[o] + ad \* (r - l + 1);  
 minm = min(minm, minv[o] + ad);  
 maxm = max(maxm, maxv[o] + ad);  
 } else {  
 int mid = l + (r - l) / 2;  
 if(ql <= mid) query(o \* 2, l, mid, ql, qr, ad + addv[o]);  
 if(qr > mid) query(o \* 2 + 1, mid + 1, r, ql, qr, ad + addv[o]);  
 }  
}  
  
void build(int o, int l, int r) {  
 if(l == r) {  
 sumv[o] = minv[o] = maxv[o] = a[l];  
 setv[o] = -1;  
 } else {  
 int mid = l + (r - l) / 2;  
 build(o \* 2, l, mid);  
 build(o \* 2 + 1, mid + 1, r);  
 minv[o] = min(minv[o \* 2], minv[o \* 2 + 1]);  
 maxv[o] = max(maxv[o \* 2], maxv[o \* 2 + 1]);  
 sumv[o] = sumv[o \* 2] + sumv[o \* 2 + 1];  
 setv[o] = -1;  
 }  
}

**11 前缀树 + AC自动机**

struct Trie {  
 int ch[max\_node][sigma\_size], val[max\_node], sz;  
 Trie() { sz = 1; memset(ch[0], 0, sizeof(ch[0])); }  
  
 int idx(char c) { return c - 'a'; }  
  
 void insert(char\* s, int v) { // Assume v != 0.  
 int u = 0, n = strlen(s);  
 for(int i = 0; i < n; i++) {  
 int c = idx(s[i]);  
 if(!ch[u][c]) {  
 memset(ch[sz], 0, sizeof(ch[sz]));  
 val[sz] = 0;  
 ch[u][c] = sz++;  
 }  
 u = ch[u][c];  
 }  
 val[u] = v;  
 }  
  
 int query(char\* s) { // If fail, just return 0.  
 int u = 0, n = strlen(s);  
 for(int i = 0; i < n; i++) {  
 int c = idx(s[i]);  
 if(!ch[u][c]) return 0;  
 u = ch[u][c];  
 }  
 return val[u];  
 }  
  
 // Next part is AC algorithm  
 // f[j] means j has already matched  
 int f[max\_node], last[max\_node];  
 void getfail() {  
 queue<int> q;  
 f[0] = 0;  
 for(int c = 0; c < sigma\_size; c++) {  
 int u = ch[0][c];  
 if(u) { f[u] = 0; q.push(u); last[u] = 0; }  
 }  
  
 while(!q.empty()) {  
 int r = q.front(); q.pop();  
 for(int c = 0; c < sigma\_size; c++) {  
 int u = ch[r][c];  
 if(!u) continue; // M2: if(!u) { ch[r][c] = ch[f[r]][c]; continue; }  
 q.push(u);  
 int v = f[r];  
 while(v && !ch[v][c]) v = f[v];  
 f[u] = ch[v][c];  
 last[u] = val[f[u]] ? f[u] : last[f[u]];  
 }  
 }  
 }  
  
 void print(int j) {  
 if(j) {  
 printf("%d %d\n", j, val[j]);  
 print(last[j]);  
 }  
 }  
  
 void find(char\* s) {  
 int n = strlen(s), j = 0;  
 for(int i = 0; i < n; i++) {  
 int c = idx(s[i]);  
 while(j && !ch[j][c]) j = f[j]; // M2: detele this statement.  
 j = ch[j][c];  
 if(val[j]) print(j);  
 else if(last[j]) print(last[j]);  
 }  
 }  
};

**12 KMP (MP)**

// Get until f[m]. Actually to f[m-1] should be fine (i < m - 1).  
void getFail(char\* P, int\* f) {  
 int m = strlen(P);  
 f[0] = 0; f[1] = 0;  
 for(int i = 1; i < m; i++) {  
 int j = f[i];  
 while(j && P[i] != P[j]) j = f[j];  
 f[i + 1] = P[i] == P[j] ? j + 1 : 0;  
 }  
}  
  
// find P in T. len(P) <= len(T).  
void find(char\* T, char\* P, int\* f) {  
 getFail(P, f);  
 int n = strlen(T), m = strlen(P), j = 0;  
 for(int i = 0; i < n; i++) {  
 while(j && T[i] != P[j]) j = f[j];  
 if(T[i] == P[j]) j++;  
 if(j == m) printf("%d\n", i - m + 1);  
 }  
}

**13 Kosaraju**

vector<int> in[N], out[N], s;

bool vis[N];

int sccno[N], scc\_cnt;

void dfs1(int u) {  
 if(vis[u]) return;  
 vis[u] = true;  
 for(int i : out[u]) dfs1(i);  
 s.push\_back(u);  
}  
  
void dfs2(int u) {  
 if(sccno[u]) return;  
 sccno[u] = scc\_cnt;  
 for(int i : in[u]) dfs2(i);  
}

void find\_scc(int n) {  
 scc\_cnt = 0;  
 s.clear();  
 memset(sccno, 0, sizeof(sccno));  
 memset(vis, 0, sizeof(vis));  
 for(int i = 0; i < n; i++) dfs1(i);  
 for(int i = n - 1; i >= 0; i--)  
 if(!sccno[s[i]]) { scc\_cnt++; dfs2(s[i]); }  
}

**14 Tarjan**

vector<int> out[N];  
int pre[N], lowlink[N], sccno[N], dfs\_clock, scc\_cnt;  
stack<int> s;  
  
void dfs(int u) {  
 pre[u] = lowlink[u] = ++dfs\_clock;  
 s.push(u);  
 for(int i : out[u])  
 if(!pre[i]) {  
 dfs(i);  
 lowlink[u] = min(lowlink[u], lowlink[i]);  
 } else if(!sccno[i]) {  
 lowlink[u] = min(lowlink[u], pre[i]);  
 }  
 if(lowlink[u] == pre[u]) {  
 scc\_cnt++;  
 for(;;) {  
 int x = s.top(); s.pop();  
 sccno[x] = scc\_cnt;  
 if(x == u) break;  
 }  
 }  
}  
  
void find\_scc(int n) {  
 dfs\_clock = scc\_cnt = 0;  
 memset(sccno, 0, sizeof(sccno));  
 memset(pre, 0, sizeof(pre));  
 for(int i = 0; i < n; i++)  
 if(!pre[i]) dfs(i);  
}

**15后缀数组+RMQ求最长公共前缀LCP**

char s[N];  
int sa[N], t[N \* 2], t2[N \* 2], c[N], n, m;  
// sa: 最终排名情况, c: 基数排序辅助数组  
// O(nlogn)  
// 需要在最后添加一个不会出现且最小的字符，常为$  
void build\_sa(int m) { // m: 可能包含的最大字符值+1  
 memset(t, 0, sizeof(t));  
 memset(t2, 0, sizeof(t2));  
 memset(sa, 0, sizeof(sa));  
 int \*x = t, \*y = t2; // x: 存储字符的编号, y: 存储第二元的排名情况  
 // 基数排序  
 for(int i = 0; i < m; i++) c[i] = 0;  
 for(int i = 0; i < n; i++) c[x[i] = s[i]]++;  
 for(int i = 1; i < m; i++) c[i] += c[i - 1];  
 for(int i = n - 1; i >= 0; i--) sa[--c[x[i]]] = i;  
 for(int k = 1; k <= n; k <<= 1) {  
 int p = 0;  
 // 利用sa数组排序第二元  
 for(int i = n - k; i < n; i++) y[p++] = i;  
 for(int i = 0; i < n; i++)  
 if(sa[i] >= k) y[p++] = sa[i] - k;  
 // 排序第一元  
 for(int i = 0; i < m; i++) c[i] = 0;  
 for(int i = 0; i < n; i++) c[x[y[i]]]++; // 第y[i]个二元组的第一元为x[y[i]]  
 for(int i = 0; i < m; i++) c[i] += c[i - 1];  
 for(int i = n - 1; i >= 0; i--) sa[--c[x[y[i]]]] = y[i];  
 // 根据sa和y重新计算x数组  
 swap(x, y);  
 p = 1; x[sa[0]] = 0;  
 for(int i = 1; i < n; i++)  
 x[sa[i]] = y[sa[i - 1]] == y[sa[i]] && y[sa[i - 1] + k] == y[sa[i] + k] ? p - 1 : p++;  
 if(p >= n) break; // 即使倍增也不会改变  
 m = p; // 更新可能包含的字符数的最大值  
 }  
}  
  
// O(mlogn)查询, m为pattern的长度, n为s(原字符串)的长度  
int cmp\_suffix(char\* pattern, int p) { return strncmp(pattern, s + sa[p], m); }  
int find(char\* P) { // 此为找到一个匹配位置。若为找到所有，则改为upper\_bound - lower\_bound.  
 m = strlen(P); // 返回在sa中的编号  
 if(cmp\_suffix(P, 0) < 0) return -1;  
 if(cmp\_suffix(P, n - 1) > 0) return -1;  
 int l = 0, r = n;  
 while(l < r) {  
 int mid = l + (r - l) / 2;  
 int res = cmp\_suffix(P, mid);  
 if(!res) return mid;  
 if(res < 0) r = mid;  
 else l = mid + 1;  
 }  
 return -1;  
}  
  
int rank[N], height[N];  
// 下面计算最长公共前缀LCP。  
// rank[i]: 后缀i在sa数组中的下标, height[i]: sa[i - 1]和sa[i]的LCP  
// O(n)  
void getHeight() {  
 int k = 0;  
 for(int i = 0; i < n; i++) rank[sa[i]] = i;  
 for(int i = 0; i < n - 1; i++) {  
 if(k) k--;  
 int j = sa[rank[i] - 1];  
 while(s[i + k] == s[j + k]) k++;  
 height[rank[i]] = k;  
 }  
}  
  
// 下面是RMQ，对于两个后缀i和j (rank[i] < rank[j]), 其LCP等于height[rank[i] + 1]...height[rank[j]]的最小值，所以可以用RMQ求出  
int d[N][30];  
  
void RMQ\_init(int n) {  
 for(int i = 0; i < n; i++) d[i][0] = sa[i];  
 for(int j = 1; (1 << j) <= n; j++)  
 for(int i = 0; i + (1 << j) <= n; i++)  
 d[i][j] = min(d[i][j - 1], d[i + (1 << (j - 1))][j - 1]);  
}  
  
int RMQ(int l, int r) {  
 int k = 0;  
 while((1 << (k + 1)) <= r - l + 1) k++;  
 return min(d[l][k], d[r - (1 << k) + 1][k]);  
}  
  
// 求原字符串以s[i]开头和s[j]开头的后缀的最长公共前缀  
int LCP(int l, int r) {  
 l = rank[l], r = rank[r];  
 if(l > r) swap(l, r);  
 return RMQ(l + 1, r);  
}

**16割顶&桥**

// 连通图的割顶和桥  
int pre[N], dfs\_clock, low[N]; // low[u]: u及其后代能连回的最早祖先的pre值  
bool cut[N]; // cut[i]代表i是否为割顶  
vector<pair<int, int> > bridge; // bridge记录所有的桥  
vector<int> G[N];  
  
int dfs(int u, int fa) {  
 int lowu = pre[u] = ++dfs\_clock, child = 0;  
 for(int i = 0; i < G[u].size(); i++) {  
 int v = G[u][i];  
 if(!pre[v]) { // son  
 child++;  
 int lowv = dfs(v, u);  
 lowu = min(lowu, lowv);  
 if(lowv >= pre[u]) cut[u] = true;  
 if(lowv > pre[u]) bridge.push\_back(pair<int, int>(u, v));  
 } else if(pre[v] < pre[u] && v != fa) { // 反向边  
 lowu = min(lowu, pre[v]);  
 }  
 }  
 if(fa < 0 && child == 1) cut[u] = false;  
 low[u] = lowu;  
 return lowu;  
}

**17 点双连通分量**

struct Edge {  
 int u, v;  
};  
  
bool cut[N];  
int pre[N], bccno[N], dfs\_clock, bcc\_cnt;  
vector<int> G[N], bcc[N];  
stack<Edge> S;  
  
int dfs(int u, int fa) {  
 int lowu = pre[u] = ++dfs\_clock;  
 int child = 0;  
 for(int i = 0; i < G[u].size(); i++) {  
 int v = G[u][i];  
 Edge e = (Edge) {u, v};  
 if(!pre[v]) {  
 S.push(e);  
 child++;  
 int lowv = dfs(v, u);  
 lowu = min(lowu, lowv);  
 if(lowv >= pre[u]) {  
 cut[u] = true;  
 bcc\_cnt++; bcc[bcc\_cnt].clear();  
 for(;;) {  
 Edge x = S.top(); S.pop();  
 if(bccno[x.u] != bcc\_cnt) { bcc[bcc\_cnt].push\_back(x.u); bccno[x.u] = bcc\_cnt; }  
 if(bccno[x.v] != bcc\_cnt) { bcc[bcc\_cnt].push\_back(x.v); bccno[x.v] = bcc\_cnt; }  
 if(x.u == u && x.v == v) break;  
 }  
 }  
 } else if(pre[v] < pre[u] && v != fa) {  
 S.push(e);  
 lowu = min(lowu, pre[v]);  
 }  
 }  
 if(fa < 0 && child == 1) cut[u] = false;  
 return lowu;  
}  
  
void find\_bcc(int n) {  
 memset(pre, 0, sizeof(pre));  
 memset(cut, 0, sizeof(cut));  
 memset(bccno, 0, sizeof(bccno));  
 dfs\_clock = bcc\_cnt = 0;  
 for(int i = 0; i < n; i++)  
 if(!pre[i]) dfs(i, -1);  
}

**18边双连通分量**

struct Edge {  
 int next;  
 Edge(int next): next(next) {}  
};  
  
int pre[N], dfs\_clock, ebc\_cnt, ebcno[N];  
bool bridge[2 \* M];  
vector<Edge> edges;  
vector<int> G[N], ebc[N];  
  
int dfs(int u, int fa) {  
 int lowu = pre[u] = ++dfs\_clock;  
 for(int i = 0; i < G[u].size(); i++) {  
 int v = edges[G[u][i]].next;  
 if(!pre[v]) {  
 int lowv = dfs(v, u);  
 lowu = min(lowu, lowv);  
 if(lowv > lowu) {  
 bridge[G[u][i]] = true;  
 bridge[G[u][i] ^ 1] = true;  
 }  
 } else if(pre[v] < pre[u] && v != fa) {  
 lowu = min(lowu, pre[v]);  
 }  
 }  
 return lowu;  
}  
  
void dfs2(int u) {  
 ebcno[u] = ebc\_cnt;  
 ebc[ebc\_cnt].push\_back(u);  
 for(int i = 0; i < G[u].size(); i++) {  
 if(bridge[G[u][i]]) continue;  
 int v = edges[G[u][i]].next;  
 if(!ebcno[v]) dfs2(v);  
 }  
}

**19 2-SAT**

struct TwoSAT {  
 int n;  
 vector<int> G[N \* 2];  
 bool mark[N \* 2];  
 int S[N \* 2], c;  
  
 void init(int n) {  
 this->n = n;  
 for(int i = 0; i < n \* 2; i++) G[i].clear();  
 memset(mark, false, sizeof(mark));  
 }  
  
 void add\_clause(int x, int xval, int y, int yval) {  
 x = 2 \* x + xval;  
 y = 2 \* y + yval;  
 G[x ^ 1].push\_back(y);  
 G[y ^ 1].push\_back(x);  
 }  
  
 bool dfs(int x) {  
 if(mark[x ^ 1]) return false;  
 if(mark[x]) return true;  
 //if(mark[x ^ 1]) return false;  
 mark[x] = true;  
 S[c++] = x;  
 for(int i = 0; i < G[x].size(); i++)  
 if(!dfs(G[x][i])) return false;  
 return true;  
 }  
  
 bool solve() {  
 for(int i = 0; i < 2 \* n; i += 2)  
 if(!mark[i] && !mark[i + 1]) {  
 c = 0;  
 if(!dfs(i)) {  
 while(c) mark[S[--c]] = false;  
 if(!dfs(i + 1)) return false;  
 }  
 }  
 return true;  
 }  
};

**20 Prim O(|E|log|V|)**

struct Node {  
 int num, dis;  
 Node(int num, int dis): num(num), dis(dis) {}  
 bool operator < (const Node& rhs) const {  
 return dis > rhs.dis;  
 }  
};  
  
struct Edge {  
 int to, cost;  
 Edge(int to, int cost): to(to), cost(cost) {}  
};  
  
int mincost[N];  
vector<Edge> edges;  
vector<int> G[N];  
bool used[N];  
  
int Prim() {  
 memset(used, false, sizeof(used));  
 memset(mincost, INF, sizeof(mincost));  
 mincost[0] = 0;  
  
 priority\_queue<Node> pq;  
 pq.push(Node(0, 0));  
  
 int ans = 0;  
 while(!pq.empty()) {  
 Node u = pq.top(); pq.pop();  
 if(u.dis > mincost[u.num]) continue;  
 used[u.num] = true;  
 ans += u.dis;  
 for(int i = 0; i < G[u.num].size(); i++) {  
 Node v(edges[G[u.num][i]].to, edges[G[u.num][i]].cost);  
 if(!used[v.num] && v.dis < mincost[v.num]) {  
 pq.push(v);  
 mincost[v.num] = v.dis;  
 }  
 }  
 }  
  
 return ans;  
}

**21 Prim O(N^2)**

// O(n^2)  
int Prim() {  
 for(int i = 0; i < n; i++) {  
 mincost[i] = INF;  
 used[i] = false;  
 }  
  
 mincost[0] = 0;  
 int ans = 0;  
  
 for(;;) {  
 int v = -1;  
  
 for(int i = 0; i < n; i++)  
 if(!used[i] && (v == -1 || mincost[i] < mincost[v])) v = i;  
 if(v == -1) break;  
 used[v] = true;  
 ans += mincost[v];  
  
 for(int i = 0; i < n; i++)  
 mincost[i] = min(mincost[i], a[i][v]);  
 }  
  
 return ans;  
}

**22 Kruskal**

int u[M], v[M], w[M], r[M], n, m, fa[N], rank[N];  
  
bool cmp(const int i, const int j) { return w[i] < w[j]; }  
int find(int x) { return fa[x] == x ? x : fa[x] = find(fa[x]); }  
void unite(int x, int y) {  
 if(rank[x] < rank[y]) fa[x] = y;  
 else { fa[y] = x; if(rank[x] == rank[y]) rank[x]++; }  
}  
int Kruskal() {  
 int ans = 0;  
 // 初始化并查集和r数组  
 for(int i = 0; i < m; i++) { r[i] = i; fa[u[i]] = u[i]; fa[v[i]] = v[i]; rank[u[i]] = rank[v[i]] = 0; }  
 sort(r, r + m, cmp);  
 for(int i = 0; i < m; i++) {  
 int e = r[i];  
 int x = find(u[e]), y = find(v[e]);  
 if(x != y) { ans += w[e]; unite(x, y); }  
 }  
 return ans;  
}

**23 次小生成树**

// starts from 0!

int u[M], v[M], w[M], r[M], n, m, fa[N], rank[N], MST;  
int maxcost[N][N];  
vector<int> MG[N];  
bool in\_mst[M];  
  
bool cmp(const int i, const int j) { return w[i] < w[j]; }  
int find(int x) { return fa[x] == x ? x : fa[x] = find(fa[x]); }  
void unite(int x, int y) {  
 if(rank[x] < rank[y]) fa[x] = y;  
 else { fa[y] = x; if(rank[x] == rank[y]) rank[x]++; }  
}  
int Kruskal() {  
 memset(in\_mst, false, sizeof(in\_mst));  
 for(int i = 0; i < n; i++) MG[i].clear();  
 int ans = 0;  
 // 初始化并查集和r数组  
 for(int i = 0; i < m; i++) { r[i] = i; fa[u[i]] = u[i]; fa[v[i]] = v[i]; rank[u[i]] = rank[v[i]] = 0; }  
 sort(r, r + m, cmp);  
 for(int i = 0; i < m; i++) {  
 int e = r[i];  
 int x = find(u[e]), y = find(v[e]);  
 if(x != y) {  
 ans += w[e];  
 unite(x, y);  
 MG[u[e]].push\_back(e);  
 MG[v[e]].push\_back(e);  
 in\_mst[e] = true;  
 }  
 }  
 return ans;  
}  
  
void dfs(int cur, int papa, vector<int>& p) {  
 p.push\_back(cur);  
 for(int i = 0; i < MG[cur].size(); i++) {  
 int e = MG[cur][i];  
 int rhs = u[e];  
 if(rhs == cur) rhs = v[e];  
 if(rhs == papa) continue;  
 maxcost[cur][rhs] = maxcost[rhs][cur] = w[e];  
 for(int j = 0; j < p.size(); j++) {  
 if(p[j] == cur) continue;  
 maxcost[p[j]][rhs] = maxcost[rhs][p[j]] = max(maxcost[p[j]][cur], w[e]);  
 }  
 dfs(rhs, cur, p);  
 }  
}  
  
int SMST() {  
 memset(maxcost, 0, sizeof(maxcost));  
 vector<int> points;  
 dfs(0, -1, points);  
 int ans = INF;  
 for(int i = 0; i < m; i++)  
 if(!in\_mst[i]) ans = min(ans, MST - maxcost[u[i]][v[i]] + w[i]);  
 return ans;  
}

**24最小树形图**

struct Edge {  
 int u, v, w;  
 Edge(int u, int v, int w): u(u), v(v), w(w) {}  
};  
  
int n, m;  
vector<Edge> edges;  
  
int id[N], inw[N], pre[N], pass[N];  
// id为缩点后的编号，inw为入边的value，pre为入边的起点，pass用于判断是否在环中  
// O(nm)  
int ZhuLiu(int root) {  
 int ans = 0;  
 for(;;) {  
 for(int i = 0; i < n; i++) { // 初始化  
 inw[i] = INF;  
 pre[i] = pass[i] = id[i] = -1;  
 }  
 for(int i = 0; i < edges.size(); i++) { // 选择最小的入边  
 Edge& e = edges[i];  
 if(e.w < inw[e.v] && e.u != e.v) { inw[e.v] = e.w; pre[e.v] = e.u; }  
 }  
 inw[root] = 0; pre[root] = root;  
 for(int i = 0; i < n; i++) { // 有点无法到达  
 if(inw[i] == INF) return -1;  
 ans += inw[i];  
 }  
  
 int idx = 0; // 缩点后的编号  
 for(int i = 0; i < n; i++) { // 判断环+缩点  
 int t = i;  
 while(pass[t] == -1) { pass[t] = i; t = pre[t]; }  
 if(pass[t] != i || t == root) continue;  
 id[t] = idx;  
 for(int j = pre[t]; j != t; j = pre[j]) id[j] = idx;  
 idx++;  
 }  
 if(idx == 0) return ans;  
 for(int i = 0; i < n; i++)  
 if(id[i] == -1) id[i] = idx++;  
  
 // 重新建图  
 for(int i = 0; i < edges.size(); i++) {  
 Edge& e = edges[i];  
 e.w -= inw[e.v];  
 e.u = id[e.u];  
 e.v = id[e.v];  
 }  
 root = id[root];  
 n = idx;  
 }  
}

**25 O(nlogn)预处理，O(logn)查询的两点间最小瓶颈路**

// 得到最小生成树后转化成有根树。cost[root]为0，pa[root]为-1。  
void dfs(int cur, int last, int level) {  
 L[cur] = level;  
 pa[cur] = last;  
 for(int e : G[cur]) {  
 int rhs = u[e];  
 if(rhs == cur) rhs = v[e];  
 if(rhs != last) {  
 cost[rhs] = w[e];  
 dfs(rhs, cur, level + 1);  
 }  
 }  
}  
  
// 预处理：pa[i][j]是第i个节点的第2^j级祖先(pa[i][0]即为父节点)  
// O(nlogn)  
void preprocess() {  
 for(int i = 0; i < n; i++) {  
 anc[i][0] = pa[i]; maxcost[i][0] = cost[i];  
 for(int j = 1; (1 << j) < n; j++) anc[i][j] = -1; // 最多有n - 1级祖先，所以要严格小于n  
 }  
 for(int j = 1; (1 << j) < n; j++)  
 for(int i = 0; i < n; i++)  
 if(anc[i][j - 1] != -1) {  
 int a = anc[i][j - 1];  
 anc[i][j] = anc[a][j - 1];  
 maxcost[i][j] = max(maxcost[i][j - 1], maxcost[a][j - 1]);  
 }  
}  
  
// 可以同时求出最近公共祖先LCA  
// O(logn)  
int query(int p, int q) {  
 int log, ans = -INF;  
 if(L[p] < L[q]) swap(p, q);  
 for(log = 1; (1 << log) <= L[p]; log++); log--;  
  
 for(int i = log; i >= 0; i--)  
 if(L[p] - (1 << i) >= L[q]) {  
 ans = max(ans, maxcost[p][i]);  
 p = anc[p][i];  
 }  
 if(p == q) return ans; // LCA为p (q)  
  
 for(int i = log; i >= 0; i--)  
 if(anc[p][i] != -1 && anc[p][i] != anc[q][i]) { // 有根且不相同  
 ans = max(ans, maxcost[p][i]); p = anc[p][i];  
 ans = max(ans, maxcost[q][i]); q = anc[q][i];  
 }  
 ans = max(ans, cost[p]);  
 ans = max(ans, cost[q]);  
 return ans; // LCA 为 pa[p] (pa[q])  
}

**26最大完美匹配**

struct KM { // 需要完美匹配存在  
 int n; // 每边的个数 (In total 2n)  
 vector<int> G[N]; // 邻接表记点  
 int W[N][N]; // 权值  
 int Lx[N], Ly[N]; // 顶标  
 int left[N]; // 右边第i个点的匹配点  
 bool S[N], T[N]; // 左/右是否已经标记(是否在交替路中)，每次找一条交替路  
  
 void init(int n) {  
 this->n = n;  
 for(int i = 0; i < n; i++) G[i].clear();  
 memset(W, 0, sizeof(W));  
 }  
  
 void AddEdge(int u, int v, int w) {  
 G[u].push\_back(v);  
 W[u][v] = w;  
 }  
  
 bool match(int u) {  
 S[u] = true;  
 for(int i = 0; i < G[u].size(); i++) {  
 int v = G[u][i];  
 if(Lx[u] + Ly[v] == W[u][v] && !T[v]) {  
 T[v] = true;  
 if(left[v] == -1 || match(left[v])) {  
 left[v] = u;  
 return true;  
 }  
 }  
 }  
 return false;  
 }  
  
 void update() {  
 int a = INF;  
 for(int u = 0; u < n; u++) if(S[u])  
 for(int i = 0; i < G[u].size(); i++) {  
 int v = G[u][i];  
 if(!T[v]) a = min(a, Lx[u] + Ly[v] - W[u][v]);  
 }  
 for(int i = 0; i < n; i++) {  
 if(S[i]) Lx[i] -= a;  
 if(T[i]) Ly[i] += a;  
 }  
 }  
  
 void solve() {  
 for(int i = 0; i < n; i++) {  
 Lx[i] = \*max\_element(W[i], W[i] + n);  
 left[i] = -1;  
 Ly[i] = 0;  
 }  
 for(int u = 0; u < n; u++)  
 for(;;) {  
 for(int i = 0; i < n; i++) S[i] = T[i] = false;  
 if(match(u)) break; else update();  
 }  
 }  
};

**27二分图最大匹配 (BFS)**

vector<int> G[Maxnode];  
int num\_node, num\_left, num\_right, num\_edge;  
int check[Maxnode], match[Maxnode], pre[Maxnode];  
// check为当前节点是否在交替路中，match记录是否配对，pre记录上一节点（用于更新）  
  
int hungarian() {  
 int ans = 0;  
 memset(check, -1, sizeof(check));  
 memset(match, -1, sizeof(match));  
 for(int i = 0; i < num\_left; i++) {  
 queue<int> Q;  
 Q.push(i);  
 pre[i] = -1;  
 while(!Q.empty() && match[i] == -1) { // match[i] == -1 iff 还没有找到增广路  
 int u = Q.front(); Q.pop();  
 for(int j = 0; j < G[u].size() && match[i] == -1; j++) {  
 int v = G[u][j];  
 if(check[v] != i) {  
 check[v] = i;  
 if(match[v] != -1) {  
 Q.push(match[v]);  
 pre[match[v]] = u;  
 } else {  
 int d = u, e = v;  
 while(d != -1) {  
 int t = match[d];  
 match[d] = e;  
 match[e] = d;  
 d = pre[d];  
 e = t;  
 }  
 }  
 }  
 }  
 }  
 if(match[i] != -1) ans++;  
 }  
 return ans;  
}

**28二分图最大匹配 (dfs)**

vector<int> G[Maxnode];  
int num\_node, num\_left, num\_right, num\_edge, match[Maxnode];  
bool check[Maxnode];  
  
bool dfs(int u) {  
 check[u] = true;  
 for(int i = 0; i < G[u].size(); i++) {  
 int v = G[u][i];  
 if(!check[v]) {  
 check[v] = true;  
 if(match[v] == -1 || dfs(match[v])) {  
 match[v] = u;  
 match[u] = v;  
 return true;  
 }  
 }  
 }  
 return false;  
}  
  
int hungarian() {  
 int ans = 0;  
 memset(match, -1, sizeof(match));  
 for(int u = 0; u < num\_left; u++) {  
 fill(check, check + num\_node, false);  
 if(dfs(u)) ans++;  
 }  
 return ans;  
}

**29 Dijkstra**

struct Dijkstra {

struct Edge {

int from, to, dist;

Edge(int u, int v, int d): from(u), to(v), dist(d) {}

};

int n, m, d[maxn], p[maxn];

vector<Edge> edges;

vector<int> G[maxn];

bool done[maxn];

void init(int n) {

this->n = n;

for(int i = 0; i < n; i++) G[i].clear();

edges.clear();

}

void add\_edge(int from, int to, int dist) {

edges.push\_back(Edge(from, to, dist));

m = edges.size();

G[from].push\_back(m - 1);

}

struct HeapNode {

int d, u;

bool operator < (const HeapNode& rhs) const {

return d > rhs.d;

}

HeapNode(int a, int b): d(a), u(b) {}

};

void dijkstra(int s) {

priority\_queue<HeapNode> q;

for(int i = 0; i < n; i++) d[i] = INF;

d[s] = 0;

q.push(HeapNode(0, s));

while(!q.empty()) {

HeapNode x = q.top(); q.pop(); // (Not sure) Since elements in the queue cannot be changed, so HeapNode& cannot be used here.

int u = x.u;

if(done[u]) continue; // u.d != d[u] is also ok.

done[u] = 1;

for(int i = 0; i < G[u].size(); i++) {

Edge& e = edges[G[u][i]];

if(d[e.to] > d[u] + e.dist) {

d[e.to] = d[u] + e.dist;

p[e.to] = u;

q.push(HeapNode(d[e.to], e.to));

}

}

}

}

void print(int u, int s) {

if(u != s) print(p[u], s);

cout << u << ' ';

}

};

**30. toposort**

int c[maxn];

int topo[maxn], t, n;

bool dfs(int u) {

c[u] = -1;

for (int v = 0; v < n; v++)if (G[u][v]) {

if (c[v] < 0)return false;

else if (!c[v]&& !dfs(v))return true;

}

c[u] = 1; topo[--t] = u;

}

bool toposort() {

t = n;

memset(c, 0, sizeof(c));

for (int u = 0; u < n; u++)if (!c[u])

if (!dfs(u)) return false;

return true;

}

**31. Bellmann-ford**

struct Bellman\_Ford {

struct Edge {

int from, to, dist;

Edge(int a, int b, int c): from(a), to(b), dist(c) {}

};

int n, m, d[maxn], cnt[maxn], p[maxn];

vector<int> G[maxn];

vector<Edge> edges;

bool inq[maxn];

void init(int n) {

this->n = n;

for(int i = 0; i < n; i++) G[i].clear();

edges.clear();

}

void add\_edge(int from, int to, int dist) {

edges.push\_back(Edge(from, to, dist));

m = edges.size();

G[from].push\_back(m - 1);

}

bool bellman\_ford(int s) {

memset(inq, 0, sizeof(inq));

memset(cnt, 0, sizeof(cnt));

inq[s] = 1;

for(int i = 0; i < n; i++) d[i] = INF;

d[s] = 0;

queue<int> q;

q.push(s);

while(!q.empty()) {

int u = q.front(); q.pop();

inq[u] = 0;

for(int i = 0; i < G[u].size(); i++) {

Edge& e = edges[G[u][i]];

if(d[e.to] > d[u] + e.dist) {

d[e.to] = d[u] + e.dist;

p[e.to] = u;

if(!inq[e.to]) { inq[e.to] = 1; q.push(e.to); if(++cnt[e.to] > n) return false; }

}

}

}

return true;

}

};

**32. Stable marriage problem**

**const int** N = 1000 + 10;  
  
**int** prefer[N][N], order[N][N], next[N], n;  
**int** husband[N], wife[N];  
queue<**int**> q;  
  
**void** engage(**int** man, **int** woman) {  
 **int** m = husband[woman];  
 **if**(m != -1) q.push(m);  
 wife[man] = woman;  
 husband[woman] = man;  
}  
  
**int** main() {  
 **int** T;  
 cin >> T;  
 **while**(T--) {  
 cin >> n;  
 **for**(**int** i = 0; i < n; i++) {  
 **for**(**int** j = 0; j < n; j++) { cin >> prefer[i][j]; prefer[i][j]--; }  
 next[i] = 0;  
 q.push(i);  
 }  
 **for**(**int** i = 0; i < n; i++) {  
 **int** x;  
 **for**(**int** j = 0; j < n; j++) {  
 cin >> x;  
 x--;  
 order[i][x] = j;  
 }  
 husband[i] = -1;  
 }  
 **while**(!q.empty()) {  
 **int** u = q.front(); q.pop();  
 **int** woman = prefer[u][next[u]++];  
 **if**(husband[woman] == -1 || order[woman][u] < order[woman][husband[woman]]) engage(u, woman);  
 **else** q.push(u);  
 }  
 **for**(**int** i = 0; i < n; i++) cout << wife[i] + 1 << '\n';  
 **if**(T) cout << '\n';  
 }  
 **return** 0;  
}

**33. 最大流最小割**

const int maxn = 100 + 10;

const int INF = 1000000000;

struct Edge {

int from, to, cap, flow;

};

bool operator < (const Edge& a, const Edge& b) {

return a.from < b.from || (a.from == b.from && a.to < b.to);

}

struct Dinic {

int n, m, s, t;

vector<Edge> edges; // 边数的两倍

vector<int> G[maxn]; // 邻接表，G[i][j]表示结点i的第j条边在e数组中的序号

bool vis[maxn]; // BFS使用

int d[maxn]; // 从起点到i的距离

int cur[maxn]; // 当前弧指针

void ClearAll(int n) {

for(int i = 0; i < n; i++) G[i].clear();

edges.clear();

}

void ClearFlow() {

for(int i = 0; i < edges.size(); i++) edges[i].flow = 0;

}

void AddEdge(int from, int to, int cap) {

edges.push\_back((Edge){from, to, cap, 0});

edges.push\_back((Edge){to, from, 0, 0});

m = edges.size();

G[from].push\_back(m-2);

G[to].push\_back(m-1);

}

bool BFS() {

memset(vis, 0, sizeof(vis));

queue<int> Q;

Q.push(s);

vis[s] = 1;

d[s] = 0;

while(!Q.empty()) {

int x = Q.front(); Q.pop();

for(int i = 0; i < G[x].size(); i++) {

Edge& e = edges[G[x][i]];

if(!vis[e.to] && e.cap > e.flow) {

vis[e.to] = 1;

d[e.to] = d[x] + 1;

Q.push(e.to);

}

}

}

return vis[t];

}

int DFS(int x, int a) {

if(x == t || a == 0) return a;

int flow = 0, f;

for(int& i = cur[x]; i < G[x].size(); i++) {

Edge& e = edges[G[x][i]];

if(d[x] + 1 == d[e.to] && (f = DFS(e.to, min(a, e.cap-e.flow))) > 0) {

e.flow += f;

edges[G[x][i]^1].flow -= f;

flow += f;

a -= f;

if(a == 0) break;

}

}

return flow;

}

int Maxflow(int s, int t) {

this->s = s; this->t = t;

int flow = 0;

while(BFS()) {

memset(cur, 0, sizeof(cur));

flow += DFS(s, INF);

}

return flow;

}

vector<int> Mincut() { // call this after maxflow

BFS();

vector<int> ans;

for(int i = 0; i < edges.size(); i++) {

Edge& e = edges[i];

if(vis[e.from] && !vis[e.to] && e.cap > 0) ans.push\_back(i);

}

return ans;

}

};

**35. 费用流**

const int maxn = 202 + 10;

const int INF = 1000000000;

typedef long long LL;

struct Edge {

int from, to, cap, flow, cost;

};

struct MCMF {

int n, m, s, t;

vector<Edge> edges;

vector<int> G[maxn];

int inq[maxn]; // 是否在队列中

int d[maxn]; // Bellman-Ford

int p[maxn]; // 上一条弧

int a[maxn]; // 可改进量

void init(int n) {

this->n = n;

for(int i = 0; i < n; i++) G[i].clear();

edges.clear();

}

void AddEdge(int from, int to, int cap, int cost) {

edges.push\_back((Edge){from, to, cap, 0, cost});

edges.push\_back((Edge){to, from, 0, 0, -cost});

m = edges.size();

G[from].push\_back(m-2);

G[to].push\_back(m-1);

}

bool BellmanFord(int s, int t, LL& ans) {

for(int i = 0; i < n; i++) d[i] = INF;

memset(inq, 0, sizeof(inq));

d[s] = 0; inq[s] = 1; p[s] = 0; a[s] = INF;

queue<int> Q;

Q.push(s);

while(!Q.empty()) {

int u = Q.front(); Q.pop();

inq[u] = 0;

for(int i = 0; i < G[u].size(); i++) {

Edge& e = edges[G[u][i]];

if(e.cap > e.flow && d[e.to] > d[u] + e.cost) {

d[e.to] = d[u] + e.cost;

p[e.to] = G[u][i];

a[e.to] = min(a[u], e.cap - e.flow);

if(!inq[e.to]) { Q.push(e.to); inq[e.to] = 1; }

}

}

}

if(d[t] > 0) return false;

ans += (LL)d[t] \* (LL)a[t];

int u = t;

while(u != s) {

edges[p[u]].flow += a[t];

edges[p[u]^1].flow -= a[t];

u = edges[p[u]].from;

}

return true;

}

// 需要保证初始网络中没有负权圈

LL Mincost(int s, int t) {

LL cost = 0;

while(BellmanFord(s, t, cost));

return cost;

}

};

**36. 平面几何**

// Geometry.cpp

#include <bits/stdc++.h>

#define LL long long

#define lson l, m, rt<<1

#define rson m+1, r, rt<<1|1

#define PI 3.1415926535897932384626

#define EXIT exit(0);

#define DEBUG puts("Here is a BUG");

#define CLEAR(name, init) memset(name, init, sizeof(name))

const double eps = 1e-6;

const int MAXN = (int)1e9 + 5;

using namespace std;

#define Vector Point

#define ChongHe 0

#define NeiHan 1

#define NeiQie 2

#define INTERSECTING 3

#define WaiQie 4

#define XiangLi 5

double torad(double deg) { return deg/180 \* PI; }

int dcmp(double x) { return fabs(x) < eps ? 0 : (x < 0 ? -1 : 1); }

struct Point {

double x, y;

Point(const Point& rhs): x(rhs.x), y(rhs.y) { } //拷贝构造函数

Point(double x = 0.0, double y = 0.0): x(x), y(y) { } //构造函数

friend istream& operator >> (istream& in, Point& P) { return in >> P.x >> P.y; }

friend ostream& operator << (ostream& out, const Point& P) { return out << P.x << ' ' << P.y; }

friend Vector operator + (const Vector& A, const Vector& B) { return Vector(A.x+B.x, A.y+B.y); }

friend Vector operator - (const Point& A, const Point& B) { return Vector(A.x-B.x, A.y-B.y); }

friend Vector operator \* (const Vector& A, const double& p) { return Vector(A.x\*p, A.y\*p); }

friend Vector operator / (const Vector& A, const double& p) { return Vector(A.x/p, A.y/p); }

friend bool operator == (const Point& A, const Point& B) { return dcmp(A.x-B.x) == 0 && dcmp(A.y-B.y) == 0; }

friend bool operator < (const Point& A, const Point& B) { return A.x < B.x || (A.x == B.x && A.y < B.y); }

void in(void) { scanf("%lf%lf", &x, &y); }

void out(void) { printf("%lf %lf", x, y); }

};

struct Line {

Point P; //直线上一点

Vector dir; //方向向量(半平面交中该向量左侧表示相应的半平面)

double ang; //极角，即从x正半轴旋转到向量dir所需要的角（弧度）

Line() { } //构造函数

Line(const Line& L): P(L.P), dir(L.dir), ang(L.ang) { }

Line(const Point& P, const Vector& dir): P(P), dir(dir) { ang = atan2(dir.y, dir.x); }

bool operator < (const Line& L) const { //极角排序

return ang < L.ang;

}

Point point(double t) { return P + dir\*t; }

};

typedef vector<Point> Polygon;

struct Circle {

Point c; //圆心

double r; //半径

Circle() { }

Circle(const Circle& rhs): c(rhs.c), r(rhs.r) { }

Circle(const Point& c, const double& r): c(c), r(r) { }

Point point(double ang) const { return Point(c.x + cos(ang)\*r, c.y + sin(ang)\*r); } //圆心角所对应的点

double area(void) const { return PI \* r \* r; }

};

double Dot(const Vector& A, const Vector& B) { return A.x\*B.x + A.y\*B.y; } //点积

double Length(const Vector& A){ return sqrt(Dot(A, A)); }

double Angle(const Vector& A, const Vector& B) { return acos(Dot(A, B)/Length(A)/Length(B)); } //向量夹角

double Cross(const Vector& A, const Vector& B) { return A.x\*B.y - A.y\*B.x; } //叉积

double Area(const Point& A, const Point& B, const Point& C) { return fabs(Cross(B-A, C-A)); }

double Dist2(const Point& A, const Point& B) { return (A.x-B.x)\*(A.x-B.x) + (A.y-B.y)\*(A.y-B.y); }

//三边构成三角形的判定

bool check\_length(double a, double b, double c) {

return dcmp(a+b-c) > 0 && dcmp(fabs(a-b)-c) < 0;

}

bool isTriangle(double a, double b, double c) {

return check\_length(a, b, c) && check\_length(a, c, b) && check\_length(b, c, a);

}

//平行四边形的判定（保证四边形顶点按顺序给出）

bool isParallelogram(Polygon p) {

if (dcmp(Length(p[0]-p[1]) - Length(p[2]-p[3])) || dcmp(Length(p[0]-p[3]) - Length(p[2]-p[1]))) return false;

Line a = Line(p[0], p[1]-p[0]);

Line b = Line(p[1], p[2]-p[1]);

Line c = Line(p[3], p[2]-p[3]);

Line d = Line(p[0], p[3]-p[0]);

return dcmp(a.ang - c.ang) == 0 && dcmp(b.ang - d.ang) == 0;

}

//梯形的判定

bool isTrapezium(Polygon p) {

Line a = Line(p[0], p[1]-p[0]);

Line b = Line(p[1], p[2]-p[1]);

Line c = Line(p[3], p[2]-p[3]);

Line d = Line(p[0], p[3]-p[0]);

return (dcmp(a.ang - c.ang) == 0 && dcmp(b.ang - d.ang)) || (dcmp(a.ang - c.ang) && dcmp(b.ang - d.ang) == 0);

}

//菱形的判定

bool isRhombus(Polygon p) {

if (!isParallelogram(p)) return false;

return dcmp(Length(p[1]-p[0]) - Length(p[2]-p[1])) == 0;

}

//矩形的判定

bool isRectangle(Polygon p) {

if (!isParallelogram(p)) return false;

return dcmp(Length(p[2]-p[0]) - Length(p[3]-p[1])) == 0;

}

//正方形的判定

bool isSquare(Polygon p) {

return isRectangle(p) && isRhombus(p);

}

//三点共线的判定

bool isCollinear(Point A, Point B, Point C) {

return dcmp(Cross(B-A, C-B)) == 0;

}

//向量绕起点旋转

Vector Rotate(const Vector& A, const double& rad) { return Vector(A.x\*cos(rad)-A.y\*sin(rad), A.x\*sin(rad)+A.y\*cos(rad)); }

//向量的单位法线(调用前请确保A 不是零向量)

Vector Normal(const Vector& A) {

double len = Length(A);

return Vector(-A.y / len, A.x / len);

}

//两直线交点(用前确保两直线有唯一交点，当且仅当Cross(A.dir, B.dir)非0)

Point GetLineIntersection(const Line& A, const Line& B) {

Vector u = A.P - B.P;

double t = Cross(B.dir, u) / Cross(A.dir, B.dir);

return A.P + A.dir\*t;

}

//点到直线距离

double DistanceToLine(const Point& P, const Line& L) {

Vector v1 = L.dir, v2 = P - L.P;

return fabs(Cross(v1, v2)) / Length(v1);

}

//点到线段距离

double DistanceToSegment(const Point& P, const Point& A, const Point& B) {

if (A == B) return Length(P - A);

Vector v1 = B - A, v2 = P - A, v3 = P - B;

if (dcmp(Dot(v1, v2)) < 0) return Length(v2);

if (dcmp(Dot(v1, v3)) > 0) return Length(v3);

return fabs(Cross(v1, v2)) / Length(v1);

}

//点在直线上的投影

Point GetLineProjection(const Point& P, const Line& L) { return L.P + L.dir\*(Dot(L.dir, P - L.P)/Dot(L.dir, L.dir)); }

//点在线段上的判定

bool isOnSegment(const Point& P, const Point& A, const Point& B) {

//若允许点与端点重合，可关闭下面的注释

//if (P == A || P == B) return true;

// return dcmp(Cross(A-P, B-P)) == 0 && dcmp(Dot(A-P, B-P)) < 0;

return dcmp(Length(P-A) + Length(B-P) - Length(A-B)) == 0;

}

//线段相交判定

bool SegmentProperIntersection(const Point& a1, const Point& a2, const Point& b1, const Point& b2) {

//若允许在端点处相交，可适当关闭下面的注释

//if (isOnSegment(a1, b1, b2) || isOnSegment(a2, b1, b2) || isOnSegment(b1, a1, a2) || isOnSegment(b2, a1, a2)) return true;

double c1 = Cross(a2-a1, b1-a1), c2 = Cross(a2-a1, b2-a1);

double c3 = Cross(b2-b1, a1-b1), c4 = Cross(b2-b1, a2-b1);

return dcmp(c1)\*dcmp(c2) < 0 && dcmp(c3)\*dcmp(c4) < 0;

}

//多边形的有向面积

double PolygonArea(Polygon po) {

int n = po.size();

double area = 0.0;

for(int i = 1; i < n-1; i++) {

area += Cross(po[i]-po[0], po[i+1]-po[0]);

}

return area \* 0.5;

}

//点在多边形内的判定(多边形顶点需按逆时针排列)

bool isInPolygon(const Point& p, const Polygon& poly) {

int n = poly.size();

for(int i = 0; i < n; i++) {

//若允许点在多边形边上，可关闭下行注释

// if (isOnSegment(p, poly[(i+1)%n], poly[i])) return true;

if (Cross(poly[(i+1)%n]-poly[i], p-poly[i]) < 0) return false;

}

return true;

}

//过定点作圆的切线

int getTangents(const Point& P, const Circle& C, std::vector<Line>& L) {

Vector u = C.c - P;

double dis = Length(u);

if (dcmp(dis - C.r) < 0) return 0;

if (dcmp(dis - C.r) == 0) {

L.push\_back(Line(P, Rotate(u, PI / 2.0)));

return 1;

}

double ang = asin(C.r / dis);

L.push\_back(Line(P, Rotate(u, ang)));

L.push\_back(Line(P, Rotate(u, -ang)));

return 2;

}

//直线和圆的交点

int GetLineCircleIntersection(Line& L, const Circle& C, vector<Point>& sol) {

double t1, t2;

double a = L.dir.x, b = L.P.x - C.c.x, c = L.dir.y, d = L.P.y - C.c.y;

double e = a\*a + c\*c, f = 2.0\*(a\*b + c\*d), g = b\*b + d\*d - C.r\*C.r;

double delta = f\*f - 4\*e\*g; //判别式

if (dcmp(delta) < 0) return 0; //相离

if (dcmp(delta) == 0) { //相切

t1 = t2 = -f / (2 \* e);

sol.push\_back(L.point(t1));

return 1;

}

t1 = (-f - sqrt(delta)) / (2.0 \* e); sol.push\_back(L.point(t1)); // 相交

t2 = (-f + sqrt(delta)) / (2.0 \* e); sol.push\_back(L.point(t2));

return 2;

}

//两圆位置关系判定

int GetCircleLocationRelation(const Circle& A, const Circle& B) {

double d = Length(A.c-B.c);

double sum = A.r + B.r;

double sub = fabs(A.r - B.r);

if (dcmp(d) == 0) return dcmp(sub) != 0;

if (dcmp(d - sum) > 0) return XiangLi;

if (dcmp(d - sum) == 0) return WaiQie;

if (dcmp(d - sub) > 0 && dcmp(d - sum) < 0) return INTERSECTING;

if (dcmp(d - sub) == 0) return NeiQie;

if (dcmp(d - sub) < 0) return NeiHan;

}

//两圆相交的面积

double GetCircleIntersectionArea(const Circle& A, const Circle& B) {

int rel = GetCircleLocationRelation(A, B);

if (rel < INTERSECTING) return min(A.area(), B.area());

if (rel > INTERSECTING) return 0;

double dis = Length(A.c - B.c);

double ang1 = acos((A.r\*A.r + dis\*dis - B.r\*B.r) / (2.0\*A.r\*dis));

double ang2 = acos((B.r\*B.r + dis\*dis - A.r\*A.r) / (2.0\*B.r\*dis));

return ang1\*A.r\*A.r + ang2\*B.r\*B.r - A.r\*dis\*sin(ang1);

}

//凸包(Andrew算法)

//如果不希望在凸包的边上有输入点，把两个 <= 改成 <

//如果不介意点集被修改，可以改成传递引用

Polygon ConvexHull(vector<Point> p) {

//预处理，删除重复点

sort(p.begin(), p.end());

p.erase(unique(p.begin(), p.end()), p.end());

int n = p.size(), m = 0;

Polygon res(n+1);

for(int i = 0; i < n; i++) {

while(m > 1 && Cross(res[m-1]-res[m-2], p[i]-res[m-2]) <= 0) m--;

res[m++] = p[i];

}

int k = m;

for(int i = n-2; i >= 0; i--) {

while(m > k && Cross(res[m-1]-res[m-2], p[i]-res[m-2]) <= 0) m--;

res[m++] = p[i];

}

m -= n > 1;

res.resize(m);

return res;

}

// 返回点集直径的平方

double diameter2(vector<Point>& points) {

vector<Point> p = ConvexHull(points);

int n = p.size();

if(n == 1) return 0;

if(n == 2) return Dist2(p[0], p[1]);

p.push\_back(p[0]); // 免得取模

double ans = 0;

for(int u = 0, v = 1; u < n; u++) {

// 一条直线贴住边p[u]-p[u+1]

for(;;) {

// 当Area(p[u], p[u+1], p[v+1]) <= Area(p[u], p[u+1], p[v])时停止旋转

// 即Cross(p[u+1]-p[u], p[v+1]-p[u]) - Cross(p[u+1]-p[u], p[v]-p[u]) <= 0

// 根据Cross(A,B) - Cross(A,C) = Cross(A,B-C)

// 化简得Cross(p[u+1]-p[u], p[v+1]-p[v]) <= 0

double diff = Cross(p[u+1]-p[u], p[v+1]-p[v]);

if(diff <= 0) {

ans = max(ans, Dist2(p[u], p[v])); // u和v是对踵点

if(diff == 0) ans = max(ans, Dist2(p[u], p[v+1])); // diff == 0时u和v+1也是对踵点

break;

}

v = (v + 1) % n;

}

}

return ans;

}

//点P在有向直线L左边的判定(线上不算)

bool isOnLeft(const Line& L, const Point& P) {

return Cross(L.dir, P-L.P) > 0;

}

//半平面交主过程

//如果不介意点集被修改，可以改成传递引用

Polygon HalfPlaneIntersection(vector<Line> L) {

int n = L.size();

int head, rear; //双端队列的第一个元素和最后一个元素的下标

vector<Point> p(n); //p[i]为q[i]和q[i+1]的交点

vector<Line> q(n); //双端队列

Polygon ans;

sort(L.begin(), L.end()); //按极角排序

q[head=rear=0] = L[0]; //双端队列初始化为只有一个半平面L[0]

for(int i = 1; i < n; i++) {

while(head < rear && !isOnLeft(L[i], p[rear-1])) rear--;

while(head < rear && !isOnLeft(L[i], p[head])) head++;

q[++rear] = L[i];

if (fabs(Cross(q[rear].dir, q[rear-1].dir)) < eps) { //两向量平行且同向，取内侧的一个

rear--;

if (isOnLeft(q[rear], L[i].P)) q[rear] = L[i];

}

if (head < rear) p[rear-1] = GetLineIntersection(q[rear-1], q[rear]);

}

while(head < rear && !isOnLeft(q[head], p[rear-1])) rear--; //删除无用平面

if (rear - head <= 1) return ans; //空集

p[rear] = GetLineIntersection(q[rear], q[head]); //计算首尾两个半平面的交点

for(int i = head; i <= rear; i++) { //从deque复制到输出中

ans.push\_back(p[i]);

}

return ans;

}

**37. 立体几何**

#include <cstdio>

#include <cstring>

#include <algorithm>

using namespace std;

const double EPS=0.000001;

typedef struct Point3 {

double x, y, z;

Point3(double xx = 0, double yy = 0, double zz = 0): x(xx), y(yy), z(zz) {}

bool operator == (const Point3& A) const {

return x==A.x && y==A.y && z==A.z;

}

}Vector3;

Point3 read\_Point3() {

double x,y,z;

scanf("%lf%lf%lf",&x,&y,&z);

return Point3(x,y,z);

}

Vector3 operator + (const Vector3 & A, const Vector3 & B) {

return Vector3(A.x + B.x, A.y + B.y, A.z + B.z);

}

Vector3 operator - (const Point3 & A, const Point3 & B) {

return Vector3(A.x - B.x, A.y - B.y, A.z - B.z);

}

Vector3 operator \* (const Vector3 & A, double p) {

return Vector3(A.x \* p, A.y \* p, A.z \* p);

}

Vector3 operator / (const Vector3 & A, double p) {

return Vector3(A.x / p, A.y / p, A.z / p);

}

double Dot(const Vector3 & A, const Vector3 & B) {

return A.x \* B.x + A.y \* B.y + A.z \* B.z;

}

double Length(const Vector3 & A) {

return sqrt(Dot(A, A));

}

double Angle(const Vector3 & A, const Vector3 & B) {

return acos(Dot(A, B) / Length(A) / Length(B));

}

Vector3 Cross(const Vector3 & A, const Vector3 & B) {

return Vector3(A.y \* B.z - A.z \* B.y, A.z \* B.x - A.x \* B.z, A.x \* B.y - A.y \* B.x);

}

double Area2(const Point3 & A, const Point3 & B, const Point3 & C) {

return Length(Cross(B - A, C - A));

}

double Volume6(const Point3 & A, const Point3 & B, const Point3 & C, const Point3 & D) {

return Dot(D - A, Cross(B - A, C - A));

}

// 四面体的重心

Point3 Centroid(const Point3 & A, const Point3 & B, const Point3 & C, const Point3 & D) {

return (A + B + C + D) / 4.0;

}

/\*\*\*\*\*\*\*\*\*\*\*\*点线面\*\*\*\*\*\*\*\*\*\*\*\*\*/

// 点p到平面p0-n的距离。n必须为单位向量

double DistanceToPlane(const Point3 & p, const Point3 & p0, const Vector3 & n)

{

return fabs(Dot(p - p0, n)); // 如果不取绝对值，得到的是有向距离

}

// 点p在平面p0-n上的投影。n必须为单位向量

Point3 GetPlaneProjection(const Point3 & p, const Point3 & p0, const Vector3 & n)

{

return p - n \* Dot(p - p0, n);

}

//直线p1-p2 与平面p0-n的交点

Point3 LinePlaneIntersection(Point3 p1, Point3 p2, Point3 p0, Vector3 n)

{

Vector3 v= p2 - p1;

double t = (Dot(n, p0 - p1) / Dot(n, p2 - p1)); //分母为0，直线与平面平行或在平面上

return p1 + v \* t; //如果是线段 判断t是否在0~1之间

}

// 点P到直线AB的距离

double DistanceToLine(const Point3 & P, const Point3 & A, const Point3 & B)

{

Vector3 v1 = B - A, v2 = P - A;

return Length(Cross(v1, v2)) / Length(v1);

}

//点到线段的距离

double DistanceToSeg(Point3 p, Point3 a, Point3 b)

{

if(a == b)

{

return Length(p - a);

}

Vector3 v1 = b - a, v2 = p - a, v3 = p - b;

if(Dot(v1, v2) + EPS < 0)

{

return Length(v2);

}

else

{

if(Dot(v1, v3) - EPS > 0)

{

return Length(v3);

}

else

{

return Length(Cross(v1, v2)) / Length(v1);

}

}

}

//求异面直线 p1+s\*u与p2+t\*v的公垂线对应的s 如果平行|重合，返回false

bool LineDistance3D(Point3 p1, Vector3 u, Point3 p2, Vector3 v, double & s)

{

double b = Dot(u, u) \* Dot(v, v) - Dot(u, v) \* Dot(u, v);

if(abs(b) <= EPS)

{return false;}

double a = Dot(u, v) \* Dot(v, p1 - p2) - Dot(v, v) \* Dot(u, p1 - p2);

s = a / b;

return true;}

// p1和p2是否在线段a-b的同侧

bool SameSide(const Point3 & p1, const Point3 & p2, const Point3 & a, const Point3 & b)

{

return Dot(Cross(b - a, p1 - a), Cross(b - a, p2 - a)) - EPS >= 0;

}

// 点P在三角形P0, P1, P2中

bool PointInTri(const Point3 & P, const Point3 & P0, const Point3 & P1, const Point3 & P2)

{

return SameSide(P, P0, P1, P2) && SameSide(P, P1, P0, P2) && SameSide(P, P2, P0, P1);

}

// 三角形P0P1P2是否和线段AB相交

bool TriSegIntersection(const Point3 & P0, const Point3 & P1, const Point3 & P2, const Point3 & A, const Point3 & B, Point3 & P)

{

Vector3 n = Cross(P1 - P0, P2 - P0);

if(abs(Dot(n, B - A)) <= EPS)

{

return false; // 线段A-B和平面P0P1P2平行或共面

}

else // 平面A和直线P1-P2有惟一交点

{

double t = Dot(n, P0 - A) / Dot(n, B - A);

if(t + EPS < 0 || t - 1 - EPS > 0)

{

return false; // 不在线段AB上

}

P = A + (B - A) \* t; // 交点

return PointInTri(P, P0, P1, P2);

}

}

//空间两三角形是否相交

bool TriTriIntersection(Point3 \* T1, Point3 \* T2)

{

Point3 P;

for(int i = 0; i < 3; i++)

{

if(TriSegIntersection(T1[0], T1[1], T1[2], T2[i], T2[(i + 1) % 3], P))

{

return true;

}

if(TriSegIntersection(T2[0], T2[1], T2[2], T1[i], T1[(i + 1) % 3], P))

{

return true;

}

}

return false;

}

//空间两直线上最近点对 返回最近距离 点对保存在ans1 ans2中

double SegSegDistance(Point3 a1, Point3 b1, Point3 a2, Point3 b2, Point3& ans1, Point3& ans2)

{

Vector3 v1 = (a1 - b1), v2 = (a2 - b2);

Vector3 N = Cross(v1, v2);

Vector3 ab = (a1 - a2);

double ans = Dot(N, ab) / Length(N);

Point3 p1 = a1, p2 = a2;

Vector3 d1 = b1 - a1, d2 = b2 - a2;

double t1, t2;

t1 = Dot((Cross(p2 - p1, d2)), Cross(d1, d2));

t2 = Dot((Cross(p2 - p1, d1)), Cross(d1, d2));

double dd = Length((Cross(d1, d2)));

t1 /= dd \* dd;

t2 /= dd \* dd;

ans1 = (a1 + (b1 - a1) \* t1);

ans2 = (a2 + (b2 - a2) \* t2);

return fabs(ans);

}

// 判断P是否在三角形A, B, C中，并且到三条边的距离都至少为mindist。保证P, A, B, C共面

bool InsideWithMinDistance(const Point3 & P, const Point3 & A, const Point3 & B, const Point3 & C, double mindist)

{

if(!PointInTri(P, A, B, C))

{

return false;

}

if(DistanceToLine(P, A, B) < mindist)

{

return false;

}

if(DistanceToLine(P, B, C) < mindist)

{

return false;

}

if(DistanceToLine(P, C, A) < mindist)

{

return false;

}

return true;

}

// 判断P是否在凸四边形ABCD（顺时针或逆时针）中，并且到四条边的距离都至少为mindist。保证P, A, B, C, D共面

bool InsideWithMinDistance(const Point3 & P, const Point3 & A, const Point3 & B, const Point3 & C, const Point3 & D, double mindist)

{

if(!PointInTri(P, A, B, C))

{

return false;

}

if(!PointInTri(P, C, D, A))

{

return false;

}

if(DistanceToLine(P, A, B) < mindist)

{

return false;

}

if(DistanceToLine(P, B, C) < mindist)

{

return false;

}

if(DistanceToLine(P, C, D) < mindist)

{

return false;

}

if(DistanceToLine(P, D, A) < mindist)

{

return false;

}

return true;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*凸包相关问题\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

//加干扰

double rand01()

{

return rand() / (double)RAND\_MAX;

}

double randeps()

{

return (rand01() - 0.5) \* EPS;

}

Point3 add\_noise(const Point3 & p)

{

return Point3(p.x + randeps(), p.y + randeps(), p.z + randeps());

}

struct Face

{

int v[3];

Face(int a, int b, int c)

{

v[0] = a;

v[1] = b;

v[2] = c;

}

Vector3 Normal(const vector<Point3> & P) const

{

return Cross(P[v[1]] - P[v[0]], P[v[2]] - P[v[0]]);

}

// f是否能看见P[i]

int CanSee(const vector<Point3> & P, int i) const

{

return Dot(P[i] - P[v[0]], Normal(P)) > 0;

}

};

// 增量法求三维凸包

// 注意：没有考虑各种特殊情况（如四点共面）。实践中，请在调用前对输入点进行微小扰动

vector<Face> CH3D(const vector<Point3> & P)

{

int n = P.size();

vector<vector<int> > vis(n);

for(int i = 0; i < n; i++)

{

vis[i].resize(n);

}

vector<Face> cur;

cur.push\_back(Face(0, 1, 2)); // 由于已经进行扰动，前三个点不共线

cur.push\_back(Face(2, 1, 0));

for(int i = 3; i < n; i++)

{

vector<Face> next;

// 计算每条边的“左面”的可见性

for(int j = 0; j < cur.size(); j++)

{

Face & f = cur[j];

int res = f.CanSee(P, i);

if(!res)

{

next.push\_back(f);

}

for(int k = 0; k < 3; k++)

{

vis[f.v[k]][f.v[(k + 1) % 3]] = res;

}

}

for(int j = 0; j < cur.size(); j++)

for(int k = 0; k < 3; k++)

{

int a = cur[j].v[k], b = cur[j].v[(k + 1) % 3];

if(vis[a][b] != vis[b][a] && vis[a][b]) // (a,b)是分界线，左边对P[i]可见

{

next.push\_back(Face(a, b, i));

}

}

cur = next;

}

return cur;

}

struct ConvexPolyhedron

{

int n;

vector<Point3> P, P2;

vector<Face> faces;

bool read()

{

if(scanf("%d", &n) != 1)

{

return false;

}

P.resize(n);

P2.resize(n);

for(int i = 0; i < n; i++)

{

P[i] = read\_Point3();

P2[i] = add\_noise(P[i]);

}

faces = CH3D(P2);

return true;

}

//三维凸包重心

Point3 centroid()

{

Point3 C = P[0];

double totv = 0;

Point3 tot(0, 0, 0);

for(int i = 0; i < faces.size(); i++)

{

Point3 p1 = P[faces[i].v[0]], p2 = P[faces[i].v[1]], p3 = P[faces[i].v[2]];

double v = -Volume6(p1, p2, p3, C);

totv += v;

tot = tot + Centroid(p1, p2, p3, C) \* v;

}

return tot / totv;

}

//凸包重心到表面最近距离

double mindist(Point3 C)

{

double ans = 1e30;

for(int i = 0; i < faces.size(); i++)

{

Point3 p1 = P[faces[i].v[0]], p2 = P[faces[i].v[1]], p3 = P[faces[i].v[2]];

ans = min(ans, fabs(-Volume6(p1, p2, p3, C) / Area2(p1, p2, p3)));

}

return ans;

}

};

**38. 加速输入输出**

std::ios\_base::sync\_with\_stdio(false);

cin.tie(0);

**39. BigInteger**

#include<iostream>

#include<cstdio>

#include<vector>

#include<algorithm>

#include<string>

#include<cstring>

#include<cassert>

using namespace std;

struct BigInteger {

static const int BASE = 100000000;

static const int WIDTH = 8;

vector<int> s;

BigInteger& clean(){ while(!s.back() && s.size() > 1) s.pop\_back(); return \*this; }

BigInteger(long long num = 0) { \*this = num; }

BigInteger operator = (long long num) {

s.clear();

do {

s.push\_back(num % BASE);

num /= BASE;

} while(num > 0);

return \*this;

}

BigInteger operator = (const string& str) {

s.clear();

int x, len = (str.length() - 1)/WIDTH + 1;

for(int i = 0; i < len; i++) {

int end = str.length() - i\*WIDTH;

int start = max(end-WIDTH, 0);

sscanf(str.substr(start, end - start).c\_str(), "%d", &x);

s.push\_back(x);

}

return \*this;

}

bool operator < (const BigInteger& b) const {

if(s.size() != b.s.size()) return s.size() < b.s.size();

for(int i = s.size() - 1; i >= 0; i--)

if(s[i] != b.s[i]) return s[i] < b.s[i];

return false;

}

bool operator > (const BigInteger& b) const { return b < \*this; }

bool operator <= (const BigInteger& b) const { return !(b < \*this); }

bool operator >= (const BigInteger& b) const { return !(\*this < b); }

bool operator != (const BigInteger& b) const { return b < \*this || b > \*this; }

bool operator == (const BigInteger& b) const { return !(b < \*this) && !(b > \*this); }

BigInteger operator + (const BigInteger& b) const {

BigInteger c;

c.s.clear();

for(int i = 0, g = 0; ; i++) {

if(g == 0 && i >= s.size() && i >= b.s.size()) break;

int x = g;

if(i < s.size()) x += s[i];

if(i < b.s.size()) x += b.s[i];

c.s.push\_back(x % BASE);

g = x / BASE;

}

return c.clean();

}

BigInteger operator - (const BigInteger& b) const {

BigInteger c;

c.s.clear();

assert(b <= \*this);

for(int i = 0, g = 0; ; i++) {

if(g == 0 && i >= s.size() && i >= b.s.size()) break;

int x = g + s[i];

if(i < b.s.size()) x -= b.s[i];

if(x < 0) { g = -1; x += BASE; } else g = 0;

c.s.push\_back(x);

}

return c.clean();

}

BigInteger operator \* (const BigInteger& b) const {

BigInteger c;

c.s.clear();

vector<long long> v(s.size()+b.s.size(), 0);

//v.clear();

long long g = 0;

for(int i = 0; i < s.size(); i++)

for(int j = 0; j < b.s.size(); j++)

v[i+j] += (long long)s[i]\*b.s[j];

for(int i = 0; ; i++) {

if(i >= v.size() && g == 0) break;

long long x = v[i] + g;

c.s.push\_back(x % BASE);

g = x / BASE;

}

return c.clean();

}

BigInteger operator / (const BigInteger& b) const {

assert(b > 0);

if(\*this < b) return 0;

BigInteger c = \*this;

BigInteger m;

m.s.clear();

for(int i = s.size()-1; i >= 0; i--) {

m = m\*BASE + s[i];

c.s[i] = bsearch(b, m);

m = m - b\*c.s[i];

}

return c.clean();

}

BigInteger operator % (const BigInteger& b) const {

assert(b > 0);

if(\*this <= b) return \*this;

BigInteger c = \*this;

BigInteger m;

m.s.clear();

for(int i = s.size()-1; i >= 0; i--) {

m = m\*BASE + s[i];

c.s[i] = bsearch(b, m);

m = m - b\*c.s[i];

}

return m;

}

int bsearch(const BigInteger& b, const BigInteger& m) const{

int L = 0, R = BASE-1, x;

while (1) {

x = (L+R)>>1;

if (b\*x<=m) { if(b\*(x+1)>m) return x; else L = x; }

else R = x;

}

}

BigInteger& operator += (const BigInteger& b) {\*this = \*this + b; return \*this;}

BigInteger& operator -= (const BigInteger& b) {\*this = \*this - b; return \*this;}

BigInteger& operator \*= (const BigInteger& b) {\*this = \*this \* b; return \*this;}

BigInteger& operator /= (const BigInteger& b) {\*this = \*this / b; return \*this;}

BigInteger& operator %= (const BigInteger& b) {\*this = \*this % b; return \*this;}

};

ostream& operator << (ostream &out, const BigInteger& x) {

out << x.s.back();

for(int i = x.s.size()-2; i >= 0; i--) {

char buf[20];

sprintf(buf, "%08d", x.s[i]);

for(int j = 0; j < strlen(buf); j++) out << buf[j];

}

return out;

}

istream& operator >> (istream &in, BigInteger& x) {

string s;

if(!(in >> s)) return in;

x = s;

return in;

}

**40. 中国剩余定理（互质）**

const int MAXN = 20;

int a[MAXN], m[MAXN], n;

int CRT(int a[], int m[], int n)

{

int M = 1;

for (int i = 0; i < n; i++) M \*= m[i];

int ret = 0;

for (int i = 0; i < n; i++)

{

int x, y;

int tm = M / m[i];

exgcd(tm, m[i], x, y);

ret = (ret + tm \* x \* a[i]) % M;

}

return (ret + M) % M;

}

**40.5 中国剩余定理（非互质）**

const int MAXN = 1000;

int a[MAXN], m[MAXN], n;

int CRT(int a[], int m[], int n)

{

if (n == 1)

{

if (m[0] > a[0]) return a[0];

else return -1;

}

int x, y, d;

for (int i = 1; i < n; i++)

{

if (m[i] <= a[i]) return -1;

d = exgcd(m[0], m[i], x, y);

if ((a[i] - a[0]) % d != 0) return -1;

int t = m[i] / d;

x = ((a[i] - a[0]) / d \* x % t + t) % t;

a[0] = x \* m[0] + a[0];

m[0] = m[0] \* m[i] / d;

a[0] = (a[0] % m[0] + m[0]) % m[0];

}

return a[0];

}

**41.高斯消元**

/\*=====================================================

复杂度：O(N^3)

输入：a 方程组对应的矩阵

n 未知数个数

l 表示是否自由元

ans 存储答案

返回值：res = -1, 表示无解。

res = 0, 表示有解。

res > 0, 表示有无数解，res表示解空间的维数。

=======================================================\*/

const int MAXN = 10000;

double a[MAXN][MAXN] = {0}, ans[MAXN] = {0};

bool l[MAXN];

int n;

int gauss(double a[][MAXN], bool l[], double ans[], const int& n)

{

int res = 0, r = 0;

for (int i = 0; i < n; i++) l[i] = false;

for (int i = 0; i < n; i++)

{

for (int j = r; j < n; j++)

{

if (fabs(a[j][i]) > eps)

{

for (int k = i; k <= n; k++)

swap(a[j][k], a[r][k]);

break;

}

}

if (fabs(a[r][i]) < eps)

{

res++;

continue;

}

for (int j = 0; j < n; j++)

{

if (j != r && fabs(a[j][i]) > eps)

{

double tmp = a[j][i] / a[r][i];

for (int k = i; k <= n; k++)

a[j][k] -= tmp \* a[r][k];

}

}

l[i] = true, r++;

}

for (int i = r; i < n; i++) if (fabs(a[i][n]) > eps) return -1;

for (int i = 0; i < n; i++)

{

if (l[i])

{

for (int j = 0; j < n; j++)

if (fabs(a[j][i]) > eps) ans[i] = a[j][n] / a[j][i];

}

}

return res;

}

**42. 辛普森积分**

double f(double x)

{

//函数体

return ans;

}

double simpson(double a, double b)

{

double c = a + (b - a) / 2;

return (f(a) + 4 \* f(c) + f(b)) \* (b - a) / 6;

}

double asr(double a, double b, double epss, double A)

{

double c = a + (b - a) / 2;

double L = simpson(a, c) , R = simpson(c, b);

if (fabs(L + R - A) <= 15 \* epss)

return L + R + (L + R - A) / 15;

return asr(a, c, epss / 2, L) + asr(c, b, epss / 2, R);

}

double solve(double l, double r)

{

return asr(l, r, eps, simpson(l, r));

}

**43. 莫比乌斯函数**

const int N = 1005;

int sum[N], isprime[N], prime[N], mu[N], pcnt;

void GetMobius()

{

pcnt = 0;

memset(isprime, 0, sizeof(isprime));

mu[1] = 1;

for (int i = 2; i < N; i++)

{

if (!isprime[i])

{

prime[pcnt++] = i;

mu[i] = -1;

}

for (int j = 0; j < pcnt && i \* prime[j] < N; j++)

{

isprime[i \* prime[j]] = 1;

if (i % prime[j]) mu[i \* prime[j]] = -mu[i];

else

{

mu[i \* prime[j]] = 0;

break;

}

}

}

for (int i = 1; i < N; i++)

sum[i] = sum[i - 1] + mu[i];

}

//求1 <= x <= n,1 <= y <= m中 gcd(x,y) == 1的个数

long long solve(int n, int m)

{

long long res = 0;

if (n > m) swap(n, m);

for (int i = 1, last = 0; i <= n; i = last + 1)

{

last = min(n / (n / i), m / (m / i));

res += (long long)(n / i) \* (m / i) \* (sum[last] - sum[i - 1]);

}

return res;

}

**44.莫队算法**

// 快速读入: Eg. read(N);

inline void read(int &x) {

x = 0; static char c;

for (; !(c >= '0' && c <= '9'); c = getchar());

for (; c >= '0' && c <= '9'; x = x \* 10 + c - '0', c = getchar()); }

// 莫队 （不带区间修改）

// 左端点所在分块作为第一关键字 右端点大小作为第二关键字

struct Cmd { int l, r, id;

friend bool operator < (const Cmd &a, const Cmd &b) {

if (belong[a.l] == belong[b.l])

return a.r < b.r;

else return belong[a.l] < belong[b.l]; }

} cmd[maxm];

int ans[maxm], belong[maxn];

int cnt[maxk]; // cnt[i] = j 表示当前区间内有j个颜色为i的东西

inline void upd(int &now, int pos, int v) { // 更新now

// 维护now -= cnt[pos];

// cnt[pos] += v;

// now += cnt[pos]; }

inline void solve(void) {

int L = 1, R = 0; // [L,R]为当前维护好的区间

int now = 0; // now为当前区间的答案

for (int i = 1; i <= M; i++) {

for (; L < cmd[i].l; L++) upd(now, L, -1);

for (; R > cmd[i].r; R--) upd(now, R, -1);

for (; L > cmd[i].l; L--) upd(now, L - 1, 1);

for (; R < cmd[i].r; R++) upd(now, R + 1, 1);

if (cmd[i].l == cmd[i].r) {

ans[cmd[i].id] =…; continue; }

ans[cmd[i].id] = now;

} } // end of solve()

int main() {

int blocksize = sqrt(N);

for (int i = 1; i <= N; i++) // [1, N]

belong[i] = (i - 1) / blocksize + 1;

for (int i = 1; i <= M; i++) {

read(cmd[i].l), read(cmd[i].r);

cmd[i].id = i; }

sort(cmd + 1, cmd + M + 1); solve();

for (int i = 1; i <= M; i++)

printf("%d\n", ans[i]);

}